WOODCOCK WASHBURN

INTELLECTUAL PROPERTY LAW

ATLANTA • PHILADELPHIA • SEATTLE

FACSIMILE

DATE: **November 12, 2009** JOB CODE:

OFFICIAL PAPER

Please deliver this and the following pages to:

Examiner:	**Charles E. Anya**
U.S.P.T.O. Group Art Unit:	**2194**
Telecopier No.:	**571-273-3757**
U.S. Serial No.:	**10/698,762**
Client/Matter No.:	**MSFT-2748/302029.01**
Sender's Name:	**Joseph F. Oriti**
Pages to Follow:	**14**

If transmission is not complete, please call our **Philadelphia Office** at **(215) 568-3100**.

COVER MESSAGE:

OFFICIAL FACSIMILE. PLEASE DELIVER TO EXAMINER IMMEDIATELY.

Per our discussion, here is a proposed draft action response for our telephone conversation to be scheduled for November 16, 2009 at 2:00 p.m.

WOODCOCK WASHBURN LLP
A Partnership Including Professional Corporations
www.woodcock.com

DOCKET NO.: MSFT-2748/302029.01
Application No.: 10/698,762
Office Action Dated: September 23, 2009

PATENT

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Application of: **Bimal Mehta, Johannes Klein, Paul Maybee, Kevin Smith, Sahsa Dadiomov, Sanjib Saha, Lee Graber, Jean-Emile Elien, Eldar Musayev**	Confirmation No.: **2802**
Application No.: **10/698,762**	Group Art Unit: **2194**
Filing Date: **October 31, 2003**	Examiner: **Charles E. Anya**

For: **HANDLING A DELIVERY FAILURE AS A PROGRAM EXCEPTION IN A DISTRIBUTED ASYNCHRONOUS ARCHITECTURE**

Mail Stop AF
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

REPLY PURSUANT TO 37 CFR § 1.111

In response to the Official Action dated **September 23, 2009** reconsideration is respectfully requested in view of the amendments and/or remarks as indicated below:

- ☐ **Amendments to the Specification** begin on page of this paper.
- ☒ **Amendments to the Claims** are reflected in the listing of the claims which begins on page 2 of this paper.
- ☐ **Amendments to the Drawings** begin on page of this paper and include an attached replacement sheet.
- ☒ **Remarks** begin on page 9 of this paper.

This listing of claims will replace all prior versions, and listings, of claims in the application.

Listing of Claims:

1. (Previously Presented) An asynchronous messaging system for processing messages, comprising:

a processor operatively coupled to a computer readable storage medium including computer executable instructions for:

executing an instance of an automated business process, the automated business process including response processing code, the response processing code including exception handling code specifying error compensation;

executing a program manager configured to manage the instance of the automated business process;

the program manager further configured to detect when the instance of the automated business process is waiting for a response to a message, wherein a response indicates a success or failure of the message;

the program manager further configured to store, when the instance is waiting for the response, at least a part of state information associated with the instance in a database and remove the instance from active memory;

the program manager further configured to determine when the response associated with the instance has been received and the program manager further configured to restore the instance from the database into memory and pass the instance the message; and

the instance further configured to process the response using the response processing code and handle exceptions using the exception handling code within the instance.

2. (Previously Presented) The system of claim 1, wherein the exception handling code is a try-catch block.

3. (Previously Presented) The system of claim 1, wherein storing the instance takes place after a predetermined time.

4. (Canceled)

5. (Previously Presented) The system of claim 1, wherein the response is received on a port defined by the instance.

6. (Previously Presented) The system of claim 1, wherein the response is a response indicative of whether or not the message was received by an intended recipient.

7. (Previously Presented) A method for processing a message in an asynchronous architecture, comprising:

determining that a response to a message sent by an instance of software code is to be received, wherein the response indicates a success or failure of the message;

determining whether the response has been received and, if the response has not been received, storing the instance of the software code in memory, thereby suspending the instance;

receiving the response and resuming the instance; and

processing the response using response processing code within the instance according to the success or failure of the message, wherein the response processing code has failure handling functionality specifying error compensation.

8. (Original) The method of claim 7, wherein determining that the response is to be received is determined by encountering a catch block within the instance.

9. (Original) The method of claim 8, wherein processing the response comprises determining whether the response indicates a failure and, if so, processing the response using the catch block.

10. (Original) The method of claim 9, further comprising, if the response indicates a success, processing the response by way of the instance of the software code.

11. (Original) The method of claim 7, wherein storing the instance occurs after a predetermined time.

12. (Original) The method of claim 7, wherein storing the instance comprises storing the instance in a database and removing the instance from active memory.

13. (Original) The method of claim 12, wherein resuming the instance comprises removing the instance from the database and restoring the instance to active memory.

14. (Original) The method of claim 7, wherein the response is received on a port defined by the instance.

15. (Original) The method of claim 7, wherein the asynchronous architecture is implemented by way of distributed business process automation software.

16. (Original) The method of claim 7, wherein the message is to be received by a remote computer.

17. (Previously Presented) A method for processing a message in an asynchronous architecture, comprising:

encountering a catch block in an instance of running business process automation software, wherein the catch block indicates that a response to a message is to be received;

determining whether the response has been received, wherein the response indicates a success or failure of the message, and if the response has not been received, storing the instance of the software code in memory, thereby suspending the instance;

receiving the response and resuming the instance in accordance with the receipt of the response; and

processing the response using response processing code within the instance according to the success or failure of the message, wherein the response processing code has failure handling functionality specifying error compensation.

18. (Original) The method of claim 17, wherein processing the response comprises determining whether the response indicates a success or failure of the message and, if the response indicates a failure, processing the response using the catch block.

19. (Original) The method of claim 18, further comprising, if the response is indicative of a success, processing the response within the instance of the automation software and logically after the catch block.

20. (Original) The method of claim 17, wherein the response is received on a port defined by the instance.

21. (Previously Presented) A computer-readable storage medium having computer-readable instructions for performing a method for processing a message in an asynchronous architecture, the method comprising:

determining that a response to a message sent by an instance of software code is to be received, wherein the response indicates a success or failure of the message;

determining whether the response has been received and, if the response has not been received, storing the instance of the software code in memory, thereby suspending the instance;

receiving the response and resuming the instance; and processing the response using response processing code within the instance according to the success or failure of the message, wherein the response processing code has failure handling functionality specifying error compensation.

22. (Original) The computer-readable medium of claim 21, wherein determining that the response is to be received is determined by encountering a catch block within the instance.

23. (Original) The computer-readable medium of claim 22, wherein processing the response comprises determining whether the response indicates a failure and, if so, processing the response using the catch block.

24. (Original) The computer-readable medium of claim 23, further comprising, if the response indicates a success, processing the response by way of the instance of the software code.

25. (Original) The computer-readable medium of claim 21, wherein storing the instance occurs after a predetermined time.

26. (Original) The computer-readable medium of claim 21, wherein storing the instance comprises storing the instance in a database and removing the instance from active memory.

27. (Original) The computer-readable medium of claim 26, wherein resuming the instance comprises removing the instance from the database and restoring the instance to active memory.

28. (Original) The computer-readable medium of claim 21, wherein the response is received on a port defined by the instance.

29. (Original) The computer-readable medium of claim 21, wherein the asynchronous architecture is implemented by way of distributed business process automation software.

30. (Original) The computer-readable medium of claim 21, wherein the message is to be received by a remote computer.

31. (Previously Presented) A computer-readable storage medium having computer-readable instructions for performing a method for processing a message in an asynchronous architecture, the method comprising:

encountering a catch block in an instance of running business process automation software, wherein the catch block indicates that a response to a message is to be received;

determining whether the response has been received, wherein the response indicates a success or failure of the message and, if the response has not been received, storing the instance of the software code in memory, thereby suspending the instance;

receiving the response and resuming the instance in accordance with the receipt of the response; and

processing the response using response processing code within the instance according to the success or failure of the message, wherein the response processing code has failure handling functionality specifying error compensation.

32. (Previously Presented) The computer-readable storage medium of claim 31, wherein processing the response comprises determining whether the response indicates a success or failure of the message and, if the response indicates a failure, processing the response using the catch block.

33. (Previously Presented) The computer-readable storage medium of claim 32, further comprising, if the response is indicative of a success, processing the response within the instance of the automation software and logically after the catch block.

34. (Previously Presented) The computer-readable storage medium of claim 31, wherein the response is received on a port defined by the instance.

REMARKS

Claims 1-3 and 5-34 are pending. Claims 1, 7, 17, 21 and 31 are independent. Claims 1-3 and 5-34 are rejected under 35 U.S.C. § 103. No claims are amended. Reconsideration of the rejection in view of the following remarks is respectfully requested.

Rejection of Claims 1-3 and 5-34 under 35 U.S.C. § 103(a)

Claims 1-3 and 5-34 are rejected under 35 U.S.C. § 103(a) as being unpatentable over (1) U.S. Patent Application Publication No. 2003/0093500, by Khodabakchian *et al.* (hereinafter referred to as "Khodabakchian") in view of at least one of: (2) U.S. Patent No. 7,036,045, issued to Broussard (hereinafter referred to as "Broussard"); and (3) U.S. Patent Application Publication No. 2003/0204835, by Budhiraja *et al.* (hereinafter referred to as "Budhiraja"). Specifically, claims 1, 2, 5-10, 12-24 and 26-34 are rejected in view of Khodabakchian and Broussard and claims 3, 11 and 25 are rejected in view of Khodabakchian, Broussard and Budhiraja. (Office Action, pp. 2-9.) Applicants respectfully traverse the rejections.

It is respectfully submitted that the combination of references, even if proper, fails to teach or suggest the claimed subject matter. In summary, the Office Action admits Khodabakchian fails to teach or suggest the claimed subject matter because it teaches a centralized exception handler 202 (FIG. 2), but alleges that Broussard teaches an instance having exception handling code or failure handling functionality. Applicants point out in detail below that Broussard, like Khodabakchian, fails to teach an instance having exception handling code or failure handling functionality.

Khodabakchian discusses a system "handling processes that interact with one or more asynchronous systems." Khodabakchian at ¶ 0005. Khodabakchian states that, "Web service orchestration server 102 contains multiple scenarios 110(1), 110(2) and 110(3)." Khodabakchian at ¶ 0023. "A scenario is a programming abstraction of a long-running process or a collaborative process." Id. "An exception handler 202 processes exceptions that are generated by orchestration engine 200 when implementing the instructions and commands in a scenario 110.

Exceptions may include, for example, a fault generated by a web service or a notice generated as a result of a web service timeout." Khodabakchian at ¶ 0025; Figure 2. Khodabakchian indicates that failures are not processed within an instance of a process, but instead are processed by an independent "exception handler" 202 on web service orchestration server 102. Id.

The Office Action admits Khodabakchian fails to teach or suggest the claimed subject matter. (Office Action, p. 4). Contrary to the admission in the Office Action, Khodabakchian is not merely "silent" on the subject of exception handling code in an instance. Khodabakchian clearly indicates that exception handling in a separate centralized process 202 (FIG. 2) that is not present in scenarios.

The Office Action alleges that Broussard teaches what Khodabakchian fails to teach. However, it is respectfully submitted, Broussard fails to teach what the Office Action says it does. Specifically, Broussard states as follows:

> Problems often occur, however, because many applications write code that catch exceptions that perform some type of default behavior. In Java, for example, exceptions are sometimes caught generically, such as with the statement:
>
> catch (Exception e) <u>{/* do nothing */}</u>.
>
> <u>In this case, nothing is done except to consume the exception</u>. While this may work fine for cases where the exceptions are intended, it may be insufficient to handle other, perhaps unintended, exceptions such as a ClassNotFoundException. In cases where the exception is due to some user error (e.g., configuration problem, missing classpath item, etc.), the program will behave badly, and there may be no visible sign of what the problem is.

Broussard, col. 1, ll. 30-43 (<u>emphasis added</u>).

> Within application logic 50 is a piece of code containing a "try-catch" block 56 to deal with any "Exception e" errors that might occur while "Do some business logic" is executed. If an error occurs 58, an exception is created 60, and the verbose exceptions mode is checked 62. If verbose is enabled for the exception 64, then the exception is output along with its stack trace 66, and the application logic continues 52.

Broussard, col. 4, ll. 57-64.

```
Try {
Do some business logic
} catch (Exception e)
{do nothing}
```

FIG. 4 (block 56).

As stated in Broussard's background section, "do nothing" means "nothing is done except to consume the exception." Thus, nothing is done in the application logic, 50, 52, let alone JVM 18. The try-catch block 56 is not even in the application logic. The try-catch block 56 is part of JVM 18 with exception dumping system 24. Input file "App" 31 is separate and is shown as application logic 50, 52 in FIG. 4. FIG. 4 clearly shows that catching the exception and the dumping system has nothing to do with the application logic 50, 52. Furthermore, there is no error handling. Instead, steps 58-66 are only error logging, not error handling. There is no error compensation. FIG. 4 shows that error occurs 58 then exception created 60 then check mode 62 then if verbose enabled for exception 64 output exception/stack trace 66. Broussard makes quite clear that nothing is done upon catching the exception, i.e., "do nothing." Like Khodabakchian, any handling code specifying error compensation is elsewhere, not in the application logic 50.

Additionally, there is no legitimate motivation to modify Khodabakchian. Khodabakchian, like other references, handles errors in a separate process using centralized exception handler 202. The Office Action asserts that the motivation to change this so that Khodabakchian's scenarios handle exceptions is to "provid[e] a condition system with a mechanism for signaling and handling unusual conditions, including errors and warnings." This generic conclusion is clearly without merit because Khodabakchian already handles unusual conditions separately using Exception Handler 202.

The foregoing remarks apply equally well to all pending claims. Further, the Office Action admits that Khodabakchian fails to disclose storing the instance after a predetermined time as claimed in claims 3, 11 and 25, but alleges Budhiraja does. (Office Action, p. 9). It is respectfully submitted that this argument is also inaccurate.

The Office Action alleges that paragraphs 0037, 0041 and 0048 of Budhiraja teach, as claimed in claims 3, 11 and 25, "wherein storing the instance takes place after a predetermined time." Specifically, the Office Action alleges use of the word "checkpoint" teaches claims 3, 11 and 25. However, nowhere does Budhiraja teach what is claimed. Checkpointing discloses nothing about setting a predetermined time to store an instance. Here is what Budhiraja actually says:

> [0037] Business process manager 408 has the ability to persistently save (checkpoint) and recover the state of the business process objects it creates. For the particular implementation shown, business process manager 408 uses a database (data store) for this purpose. Other implementations may use different persistent storage methods.
>
> [0041] The checkpointing ability of Business process manager 408 is extended so that name and label information is included with saved state information. This means that each time that business process manager 408 saves the state of a business process object, it also saves the name and label associated with the business process model used to create that business process object.
>
> [0048] Process modeler 402 and BusinessWare server 404 are configured so that sub-process models may be labeled in the same way as business process models. Thus, there may be multiple versions of a sub-process model. Each definition may have an associated label including an "initial" and a "current" version. The checkpointing ability of Business process manager 408 is configured to include the version information for sub-process models during state saving operations. This allows business process objects to be restarted using the correct versions of their nested sub-process models.

Thus, Budhiraja does not provide the disclosure missing in Khodabakchian and Broussard as to claims 3, 11 and 25. This is an additional reason the rejection should be withdrawn against claims 3, 11 and 25.

It is respectfully submitted that the claimed subject matter is allowable over the cited references. Accordingly, Applicants respectfully request reconsideration and withdrawal of the rejection of claims 1-3 and 5-34 under 35 U.S.C. § 103(a) in view of various combinations of Khodabakchian, Broussard and Budhiraja.

DOCKET NO.: MSFT-2748/302029.01
Application No.: 10/698,762
Office Action Dated: September 23, 2009

PATENT

CONCLUSION

Amendments, made herein or previously made, are without abandonment of subject matter. Applicant expressly reserves the right to, in the pending application or any application related thereto, reintroduce any subject matter removed from the scope of claims by any amendment and introduce any subject matter not present in current or previous claims.

In view of the foregoing remarks, it is respectfully submitted that this application is in condition for allowance. Reconsideration of this application and an early Notice of Allowance are respectfully requested.

Date:

Joseph F. Oriti
Registration No. 47,835

Woodcock Washburn LLP
Cira Centre
2929 Arch Street, 12th Floor
Philadelphia, PA 19104-2891
Telephone: (215) 568-3100
Facsimile: (215) 568-3439